시뮬레이터 기본 교육 및 Control & Planning 알고리즘 개발

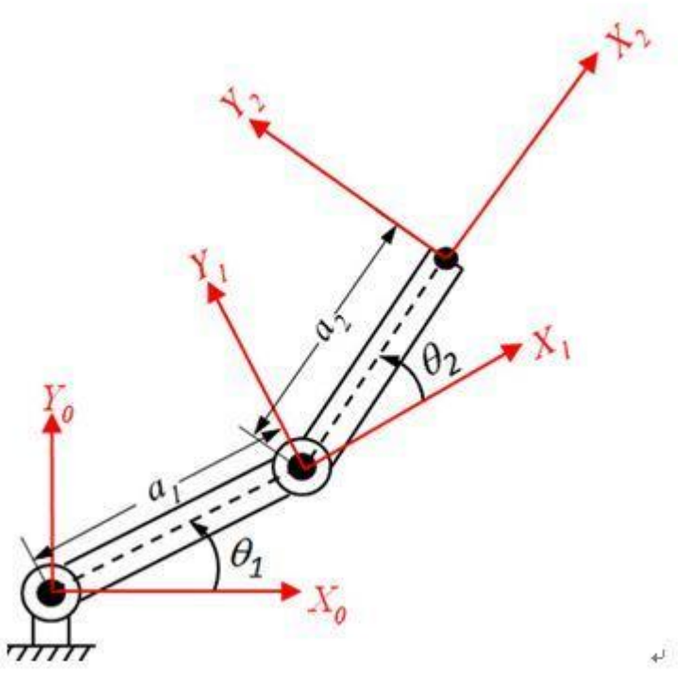
프로젝트 지향 자율주행차 전문인력 양성과정

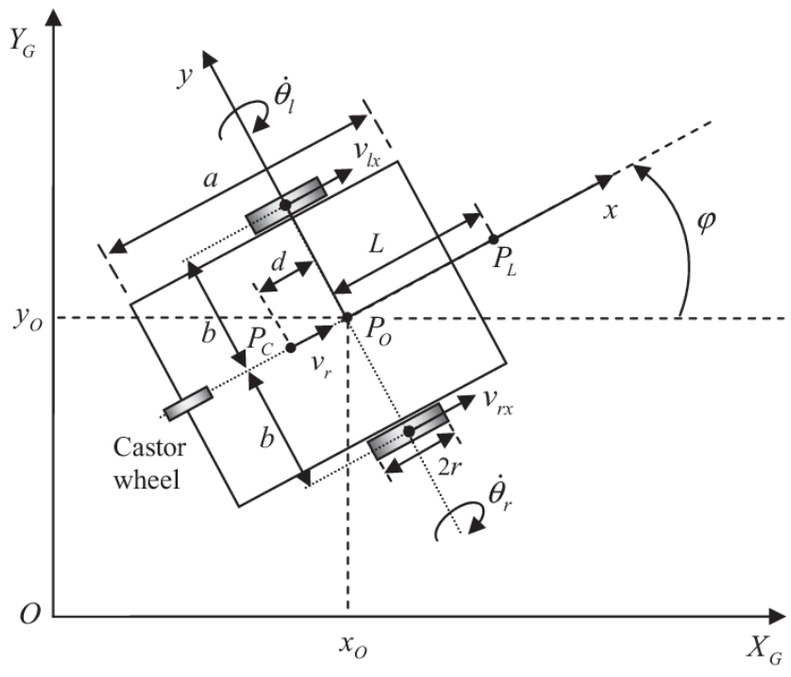
목차

1. Kinematic Model

Kinematic Model

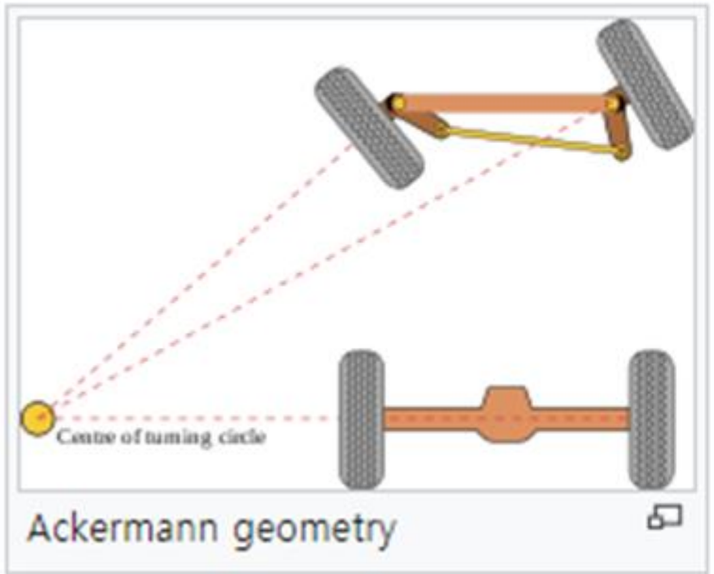
- **Kinematics**
  - **기계요소들간의 상호 연계되어 이루어지는 운동을 파악하는 방법에 대해 연구하는 학문**
  - **특정한 자세를 기하학적으로 표현한 할 수 있다. 힘을 다루는 동역학과 달리, 힘이나 토크가 고려되지 않은 움직임을 표현할 수 있다.**

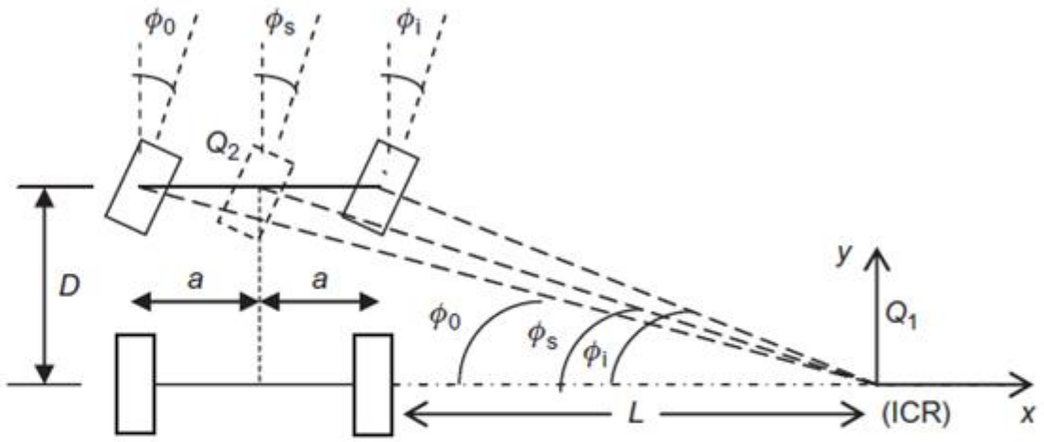




Kinematic Model

- **Ackermann geometry**
  - **자동차와 가장 흡사한 모델**





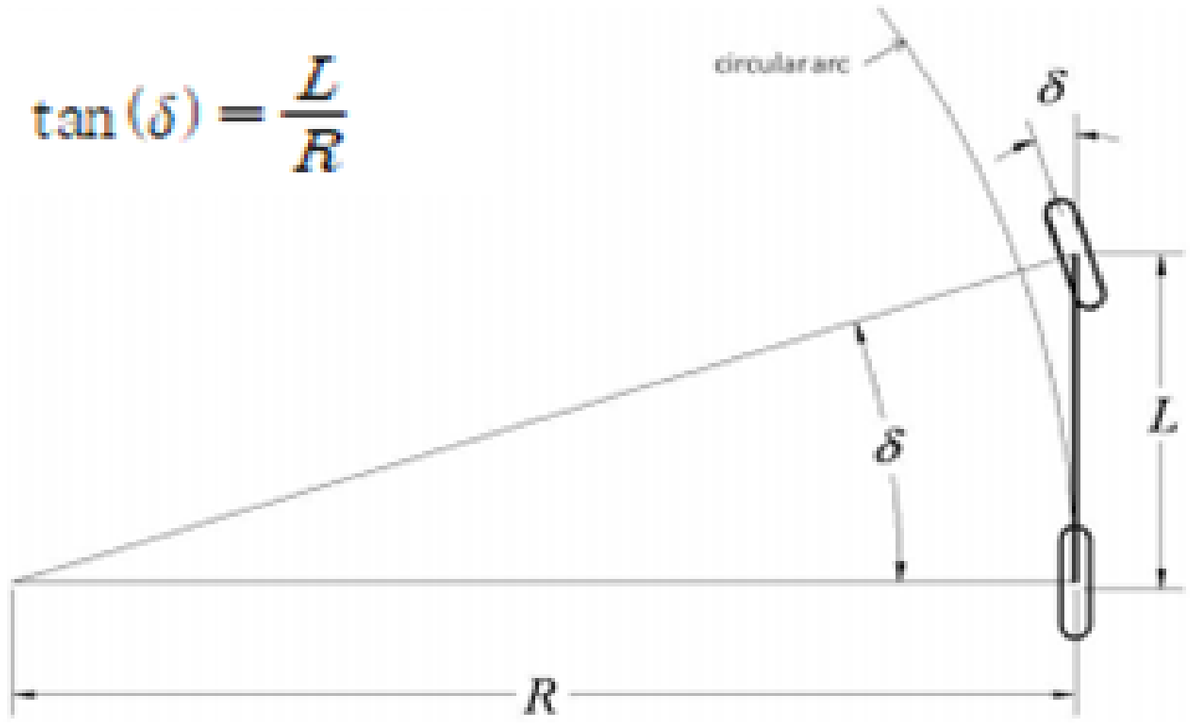
$$ctg\phi_s = (a+L)/D, \quad ctg\phi_o = (2a+L)/D, \quad ctg\phi_i = L/D$$

$$ctg\phi_s = \frac{a}{D} + ctg\phi_i \quad \text{or} \quad ctg\phi_s = ctg\phi_o - \frac{a}{D}$$

Kinematic Model

- **Bicycle geometry**
  - **조향 가능한 앞 바퀴 1개, 고정된 뒷바퀴 1개로 구성된 모델**

$$\tan(\delta) = \frac{L}{R}$$

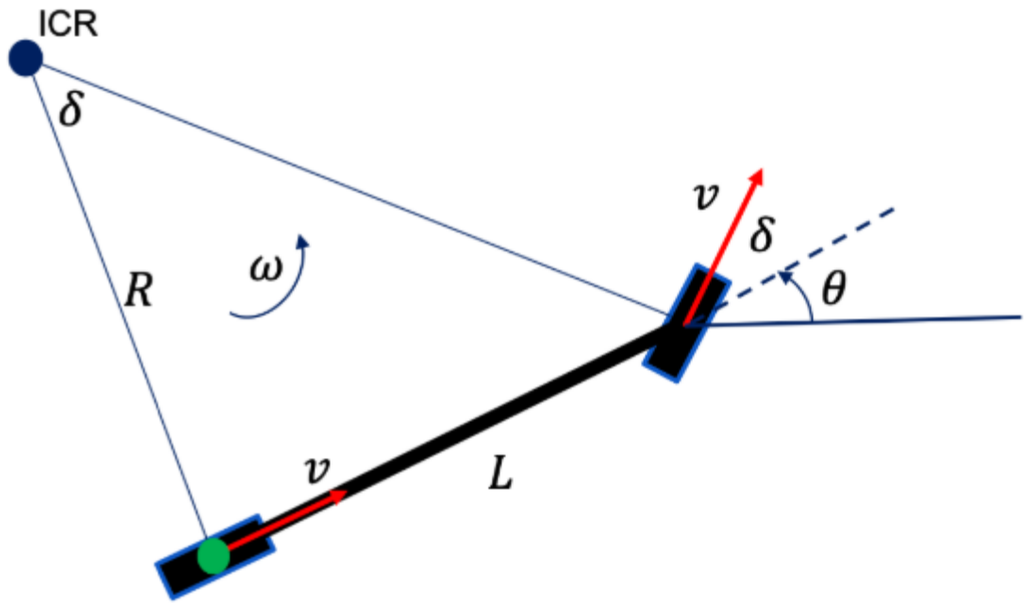


Kinematic Model

- **Bicycle kinematic Model**

  - Rear Wheel Reference Point
    - Apply Instantaneous Center of Rotation (ICR)

$$\dot{\theta} = \omega = \frac{v}{R}$$



    - Similar triangles

$$\tan\delta = \frac{L}{R}$$

    - Rotation rate equation

$$\dot{\theta} = \omega = \frac{v}{R} = \frac{v\tan\delta}{L}$$

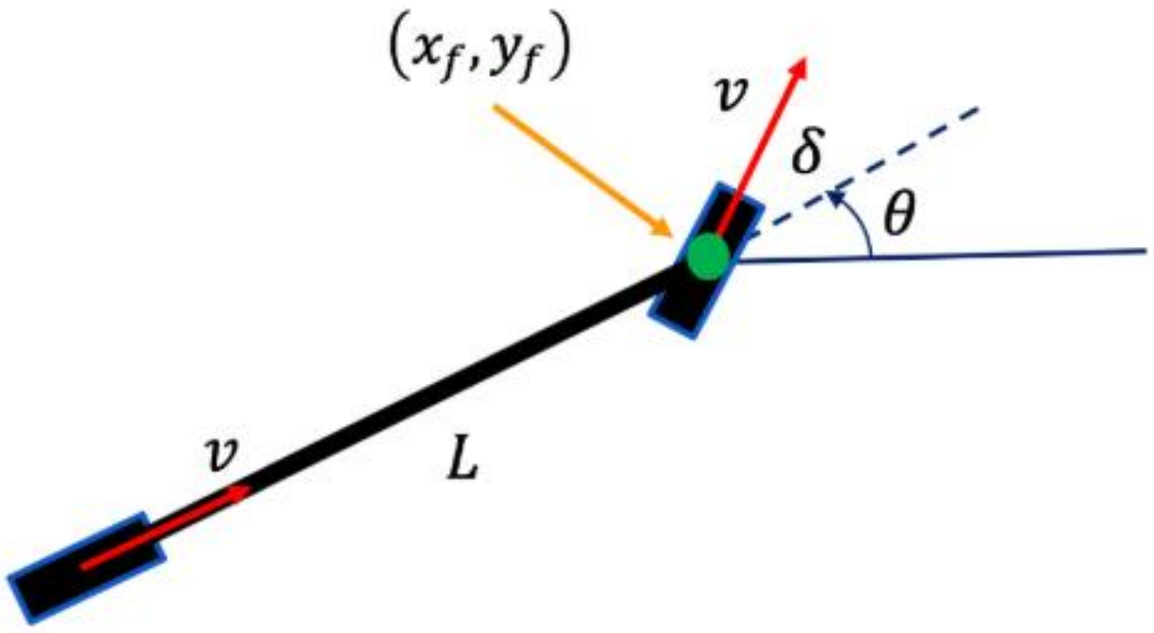
Kinematic Model

- **Bicycle kinematic Model**
  - **속도, 조향각(v, δ)을 알면 차량의 포즈(x,y,θ) 를 구할 수 있다.**

$$\dot{x}_f = v\cos(\theta + \delta)$$

$$\dot{y}_f = v\sin(\theta + \delta)$$

$$\dot{\theta} = \frac{v\sin\delta}{L}$$



Kinematic Model 실습

```python
#!/usr/bin/env python
# -*- coding: utf-8 -*-
import rospy
from sensor_msgs.msg import LaserScan,PointCloud
from std_msgs.msg import Float64
from vesc_msgs.msg import VescStateStamped
from laser_geometry import LaserProjection
from math import cos,sin,pi
from geometry_msgs.msg import Point32
from nav_msgs.msg import Odometry
import tf
from tf.transformations import euler_from_quaternion,quaternion_from_euler


class simple_kinematics :

    def __init__(self):
        rospy.init_node('simple_kinematics', anonymous=True)

        rospy.Subscriber("/sensors/core", VescStateStamped, self.status_callback)
        rospy.Subscriber("/sensors/servo_position_command", Float64, self.servo_command_callback)

        self.is_speed=False
        self.is_servo=False
        self.servo_msg=Float64()


        self.odom_pub = rospy.Publisher('/odom',Odometry, queue_size=1)
        self.odom_msg=Odometry()
        self.odom_msg.header.frame_id='/odom'

        self.rpm_gain=4614
        self.steering_angle_to_servo_gain =-1.2135
        self.steering_angle_to_servo_offset=0.5304
        self.theta=0
        self.L=0.5

        rate = rospy.Rate(20) # 20hz
```

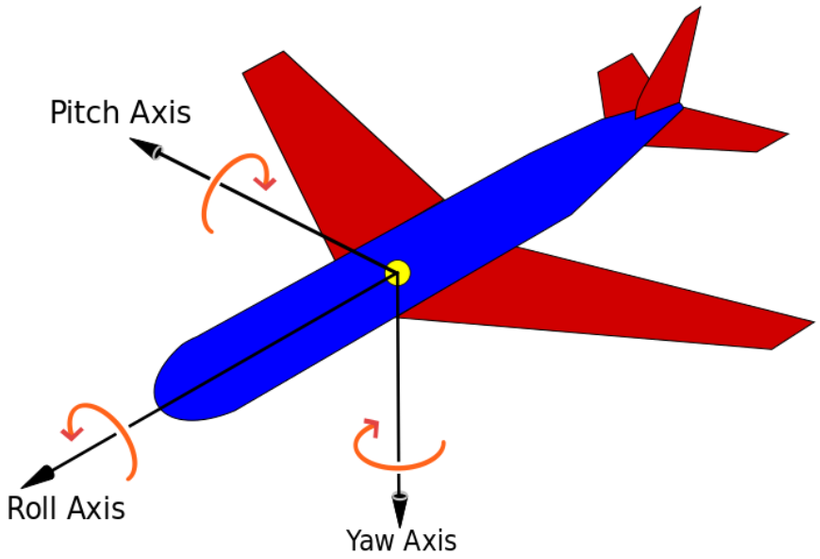
Kinematic Model 실습

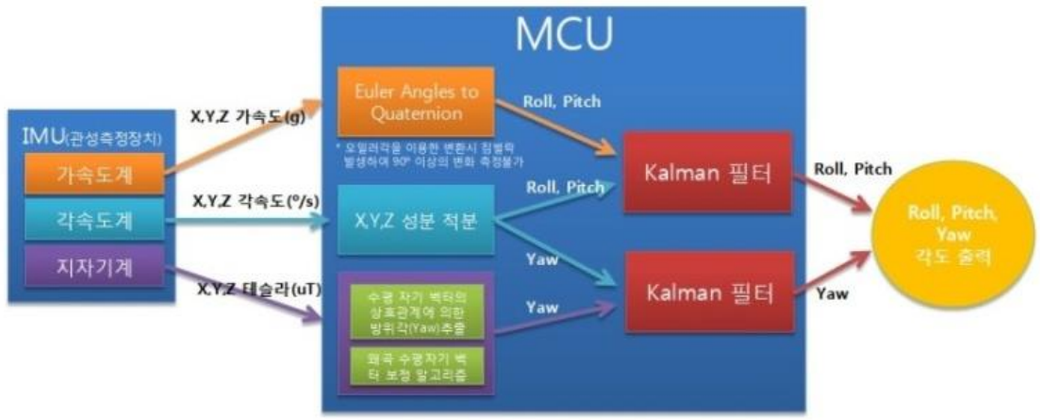
```
40          while not rospy.is_shutdown():
41
42              if self.is_servo == True and self.is_speed==True :
43                  print(self.speed,self.servo_angle_rad*180/pi)
44
45                  x_dot= self.speed*cos(self.theta+self.servo_angle_rad)/20
46                  y_dot= self.speed*sin(self.theta+self.servo_angle_rad)/20
47                  theta_dot=self.speed*sin(self.servo_angle_rad)/self.L/20
48
49                  self.odom_msg.pose.pose.position.x=self.odom_msg.pose.pose.position.x+x_dot
50                  self.odom_msg.pose.pose.position.y=self.odom_msg.pose.pose.position.y+y_dot
51                  self.theta=self.theta+theta_dot
52                  quaternion=quaternion_from_euler(0,0,self.theta)
53                  self.odom_msg.pose.pose.orientation.x=quaternion[0]
54                  self.odom_msg.pose.pose.orientation.y=quaternion[1]
55                  self.odom_msg.pose.pose.orientation.z=quaternion[2]
56                  self.odom_msg.pose.pose.orientation.w=quaternion[3]
57
58                  self.odom_pub.publish(self.odom_msg)
59                  br = tf.TransformBroadcaster()
60                  br.sendTransform(( self.odom_msg.pose.pose.position.x,  self.odom_msg.pose.pose.position.y,  self.odom_msg.pose.pose.position.z),
61                                   quaternion,
62                                   rospy.Time.now(),
63                                   "base_link",
64                                   "odom")
65              rate.sleep()
66
67
68      def status_callback(self,msg):
69          self.is_speed=True
70          rpm=msg.state.speed
71          self.speed=rpm/self.rpm_gain
72
73      def servo_command_callback(self,msg):
74          self.is_servo=True
75          servo_value=msg.data
76          self.servo_angle_rad= (servo_value-self.steering_angle_to_servo_offset)/self.steering_angle_to_servo_gain
```

2. Imu

Imu

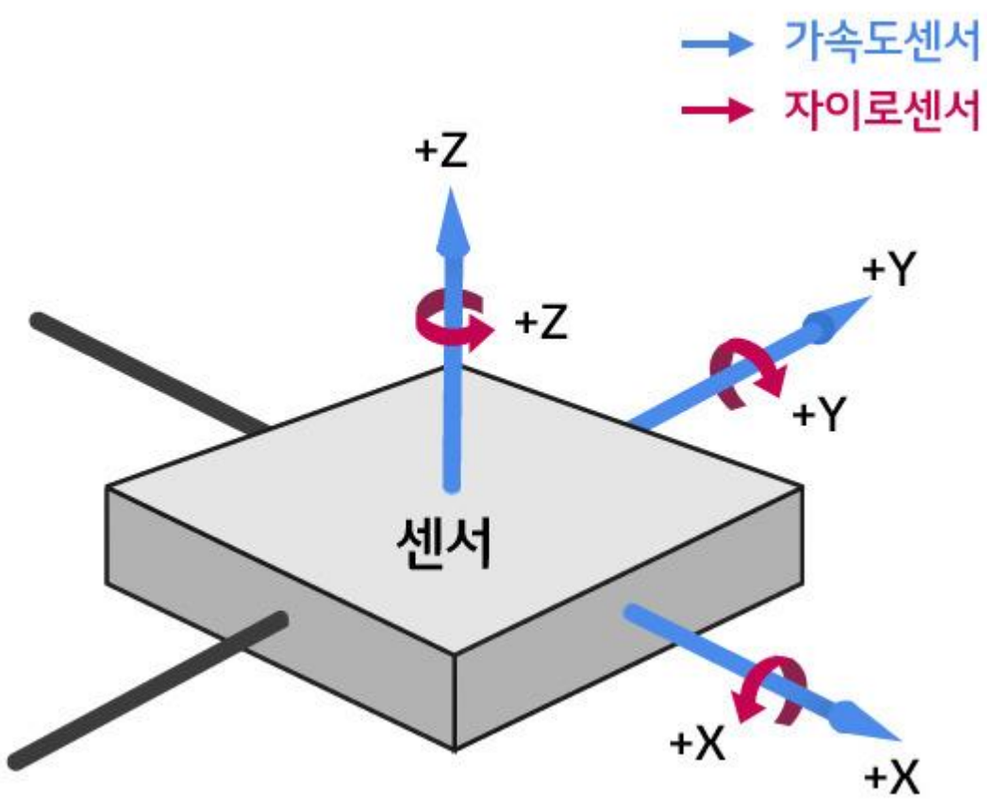
- IMU 이해
  - 관성 측정 장치로 가속도계, 회전 속도계, 자력계를 조합해서 자세를 측정함
  - 누적 에러가 생기기 때문에, 주로 다른 센서와 융합해서 사용함
  - 측정 주기가 매우 빠름

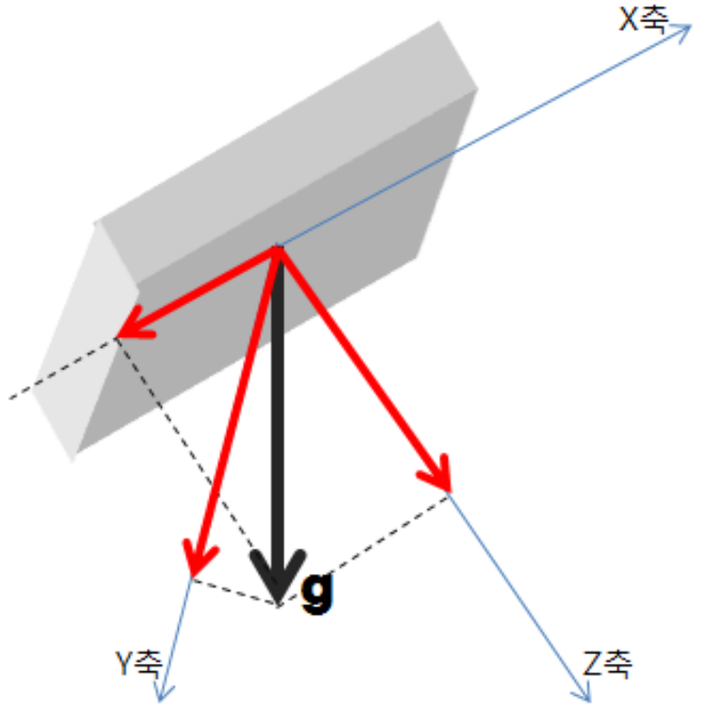




Imu

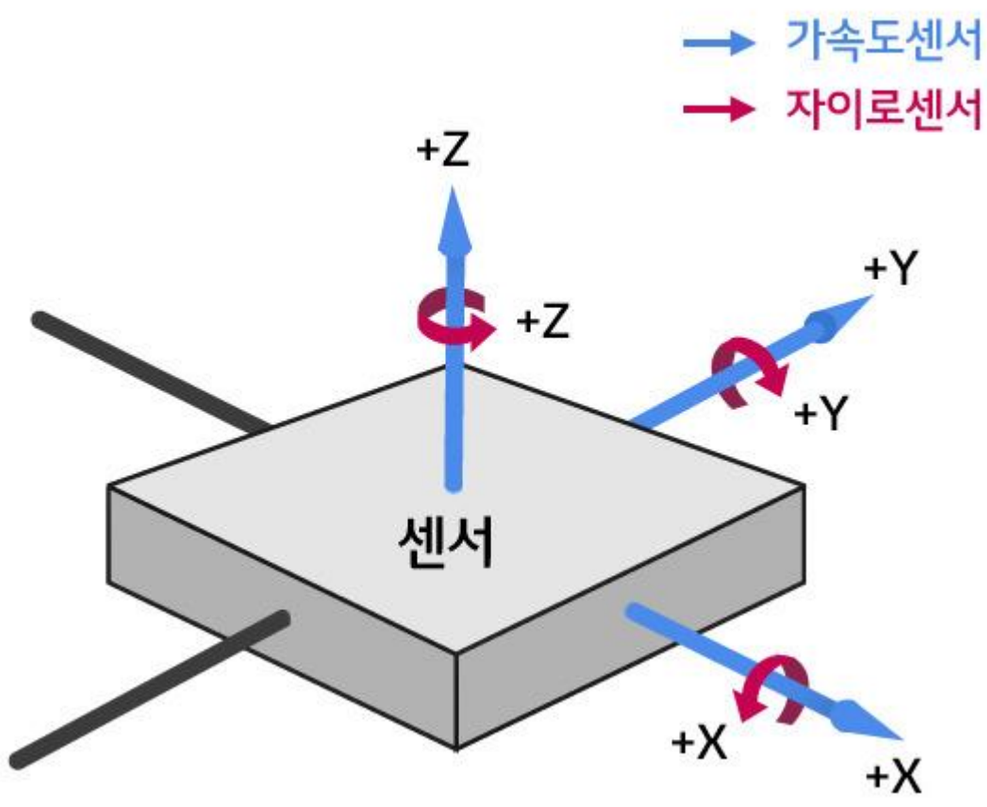
- 가속도 센서
  - 가속도 센서는 물체의 가속도와 진동, 충격 등 움직이는 힘을 측정하는 센서로 x,y,z 방향의 가속도를 측정할 수 있다.
  - 변화하는 값의 오차가 누적되지 않으나 노이즈와 이동, 진동에 취약하다는 단점이 있다. 가속도 센서 하나만으로 기울기를 정확하게 알 수 없어 이 단점을 보완하기 위해 자이로 센서를 사용한다.





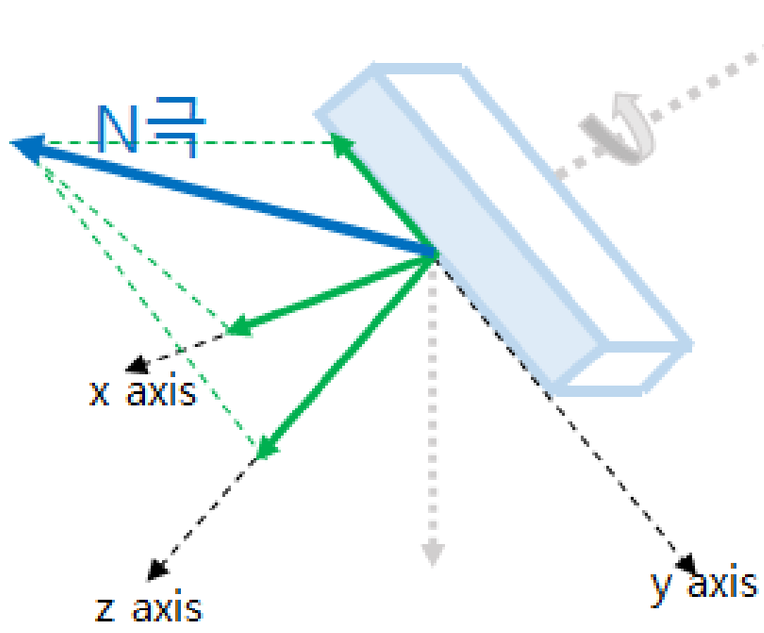
Imu

- 자이로 센서
  - 자이로 센서는 회전하는 물체의 각속도를 측정하는 센서로, 각속도에서 각도를 구하려면 전체 시간에 해당하는 만큼 적분을 통해 물체의 회전각과 기울기 등을 알 수 있다.
  - 가속도센서에 비해 자이로센서는 비교적 안정적인 값이 출력되지만 각도를 계산하는 과정에서 사용되는 적분에 의해 시간이 지날수록 누적된 오차가 발생한다.



Imu

- 지자기 센서
  - 지자기센서는 지구 자기장을 이용해 물체의 방향측정 및 방위각을 측정할 수 있는 센서이다. 이 센서는 외부의 자기신호에 민감하기 때문에 자기장이 강한 공간에서는 그 사용이 제한될 수 있으며, 3축에 대해 자기장 보정을 해야 한다는 단점이 있지만 자북이라는 절대 정보를 제공한다는 이점이 있다.



Imu

- **Euler Angle**
  - **3차원 공간의 X, Y, Z 축에 대한 회전각을 지정한 순서대로 곱해서 사용하여 arbitrary orientation을 나타낸다.**
  - **2차원 평면에서 물체의 방향을 지정해 주려면 단지 하나의 각도면 충분하지만 3차원에서는 최소한 3개의 각도 값이 필요하다.**
  - **오일러각은 특정 회전축의 조합 하나를 말하는 것이 아니라 3차원 공간에서 임의의 방향을 나타낼 수 있는 조합을 모두 가리킨다.**
  - **일반적으로 ZYX(Yaw-Pitch-Roll)회전과 XYZ 회전이 주로 사용된다.**

(y, x, z), (x, y, z), (z, x, y), ... 12가지 모두 사용 가능하다.

XYZ	XZY	XYX	XZX
YXZ	YZX	YXY	YZY
ZXY	ZYX	ZXZ	ZYZ

$$\mathbf{R}_{zyx} = \mathbf{R}_z(\psi)\mathbf{R}_y(\theta)\mathbf{R}_x(\phi)$$
$$= \begin{bmatrix} \cos\theta\cos\psi & \sin\phi\sin\theta\cos\psi - \cos\phi\sin\psi & \cos\phi\sin\theta\cos\psi + \sin\phi\sin\psi \\ \cos\theta\sin\psi & \sin\phi\sin\theta\sin\psi + \cos\phi\cos\psi & \cos\phi\sin\theta\sin\psi - \sin\phi\cos\psi \\ -\sin\theta & \sin\phi\cos\theta & \cos\phi\cos\theta \end{bmatrix}$$

$$\mathbf{R}_x(\phi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & -\sin\phi \\ 0 & \sin\phi & \cos\phi \end{bmatrix}, \quad \mathbf{R}_y(\theta) = \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix}, \quad \mathbf{R}_z(\psi) = \begin{bmatrix} \cos\psi & -\sin\psi & 0 \\ \sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Imu

- Quaternion
  - 사원수로, 복소수를 확장한 일종의 4차원 벡터
  - 세개의 축을 동시에 회전시킨다
  - 짐벌락 현상이 생기지 않음
  - 컴퓨터그래픽스에서 주로 사용

- 짐벌락 현상
  - 세 개의 축각을 이용해 항상 순서대로 회전을 하면 위와같이 한 개의 축이 쓸모 없게 된다.
  - 세 번에 나눠서 계산하기 때문에 생김

ex) x축을 90도 틀고난 후 y축으로 90도 튼것과 y축을 90도 틀고난 후 z축으로 90도 튼것과의 회전 값이 같다.

Imu

- **IMU(orientation) + Velocity**

```python
#!/usr/bin/env python
# -*- coding: utf-8 -*-

import rospy
from sensor_msgs.msg import LaserScan,PointCloud,Imu
from std_msgs.msg import Float64
from vesc_msgs.msg import VescStateStamped
from laser_geometry import LaserProjection
from math import cos,sin,pi
from geometry_msgs.msg import Point32
from nav_msgs.msg import Odometry

import tf
from tf.transformations import euler_from_quaternion,quaternion_from_euler

class imu_odom :

    def __init__(self):
        rospy.init_node('imu_odom', anonymous=True)

        rospy.Subscriber("/sensors/core", VescStateStamped, self.status_callback)
        rospy.Subscriber("/imu", Imu, self.imu_callback)

        self.is_speed=False
        self.is_imu=False

        self.odom_pub = rospy.Publisher('/odom',Odometry, queue_size=1)
        self.odom_msg=Odometry()
        self.odom_msg.header.frame_id='/odom'

        self.rpm_gain=4614
        self.theta=0

        rate = rospy.Rate(20) # 20hz
        while not rospy.is_shutdown():
```

Imu

- **IMU(orientation) + Velocity**

```
33
34          rate = rospy.Rate(20) # 20hz
35          while not rospy.is_shutdown():
36
37              if self.is_imu == True and self.is_speed==True :
38                  print(self.speed,self.theta*180/pi)
39
40                  x_dot= self.speed*cos(self.theta)/20
41                  y_dot= self.speed*sin(self.theta)/20
42
43                  self.odom_msg.pose.pose.position.x=self.odom_msg.pose.pose.position.x+x_dot
44                  self.odom_msg.pose.pose.position.y=self.odom_msg.pose.pose.position.y+y_dot
45
46                  quaternion=quaternion_from_euler(0,0,self.theta)
47                  self.odom_msg.pose.pose.orientation.x=quaternion[0]
48                  self.odom_msg.pose.pose.orientation.y=quaternion[1]
49                  self.odom_msg.pose.pose.orientation.z=quaternion[2]
50                  self.odom_msg.pose.pose.orientation.w=quaternion[3]
51
52                  self.odom_pub.publish(self.odom_msg)
53                  br = tf.TransformBroadcaster()
54                  br.sendTransform(( self.odom_msg.pose.pose.position.x,  self.odom_msg.pose.pose.position.y,  self.odom_msg.pose.pose.position.z),
55                      quaternion,
56                      rospy.Time.now(),
57                      "base_link",
58                      "odom")
59
60              rate.sleep()
61
62      def status_callback(self,msg):
63          self.is_speed=True
64          rpm=msg.state.speed
65          self.speed=rpm/self.rpm_gain
66
67      def imu_callback(self,msg):
```

Imu

- **IMU(orientation) + Velocity**

```
67      def imu_callback(self,msg):
68          imu_quaternion=(msg.orientation.x,msg.orientation.y,msg.orientation.z,msg.orientation.w)
69          if self.is_imu == False :
70              self.is_imu=True
71              _,_,self.theta_offset=euler_from_quaternion(imu_quaternion)
72          else :
73              _,_,raw_theta=euler_from_quaternion(imu_quaternion)
74              self.theta=raw_theta-self.theta_offset
75
76
77  if __name__ == '__main__':
78      try:
79          test_track=imu_odom()
80      except rospy.ROSInterruptException:
81          pass
```

3. 실습

실습

- Imu의 orientation을 사용하지 않고 odom 추정해보자
  - 방법 1. 각 속도를 적분해서 theta를 구하고, velocity를 이용해서 추정
  - 방법 2. imu만 이용해서 odom 추정하기

END